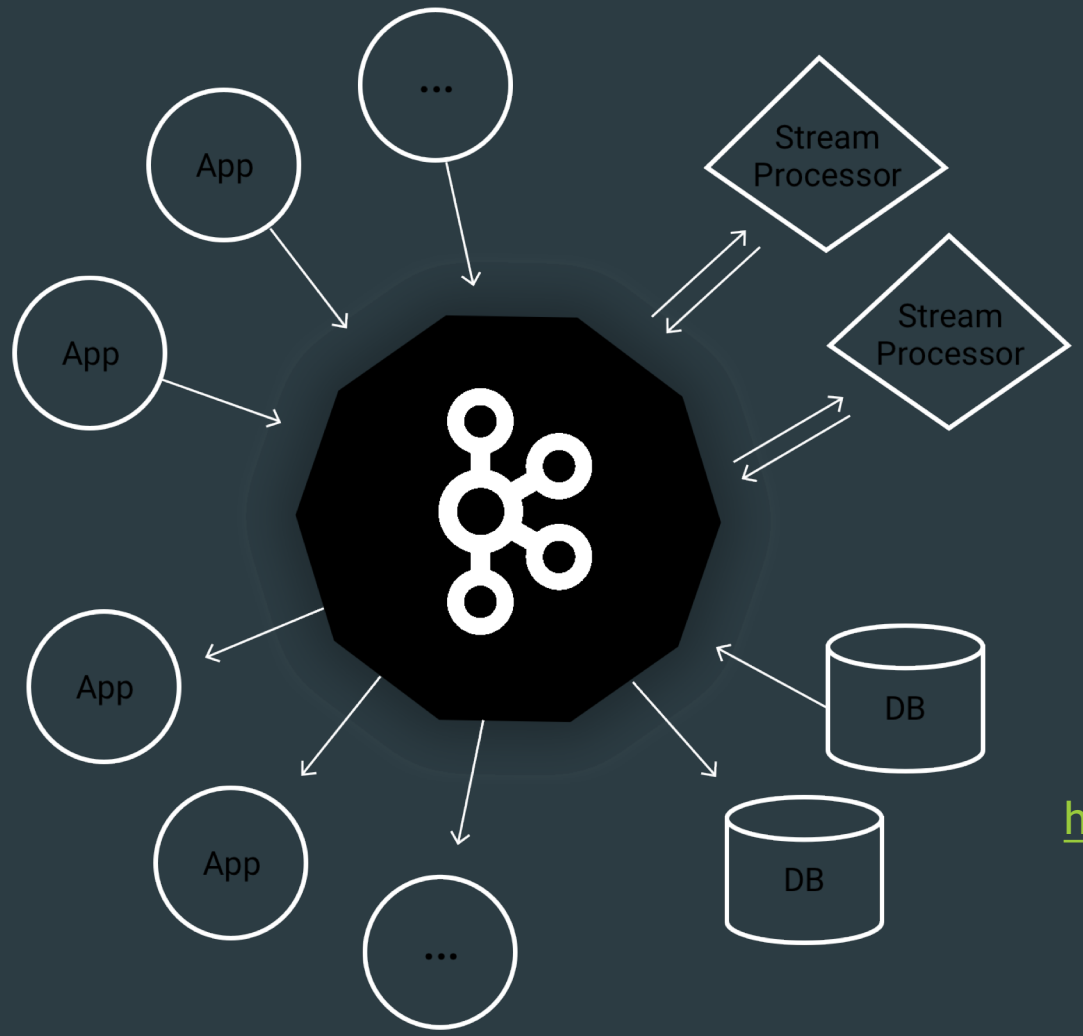


Kafka

Big data pipeline

Expliqué par David Gilson
https://github.com/gilsdav

Assets

- https://github.com/gilsdav/aw-kafka

Sommaire

- Qu’est-ce que Kafka ?
  - Qu’est-ce qu’un « Messaging System » ?
  - Cas d’utilisation
- Fondamentaux
- Intégrations existantes / Confluent Hub
- Comparaisons avec d’autres « Messaging System »
  - Kafka vs RabbitMQ
  - Azure services
    - Azure Event Hubs pour Apache Kafka
- Architecture en micro-services
- Démo

1) Qu’est-ce que Kafka ?

Qu’est-ce que Kafka ?

- Apache Kafka a été créé par LinkedIn et est devenu open-source en 2011 puis repris par Apache en 2012

- Une plateforme de **streaming** distribuée
- Capacités :
  - **Publier/Souscrire à un stream de messages** (de la même façon qu’une message queue ou qu’un messaging system)
  - Stocker des streams de donnée de façon durable et tolèrent aux pannes
  - Traiter les données du stream dans l’ordre d’émission

Qu’est-ce que Kafka ?

- Cas d’utilisation généraux:
  - Real-time streaming data pipelines: permet de transférer des données de manière fiable entre systèmes et/ou applications
  - Real-time streaming applications: des applications qui transforme ou réagissent aux flux de données

- Fonctionnement
  - Kafka tourne en tant que cluster sur un ou plusieurs serveurs pouvant s’étendre sur plusieurs Datacenters
  - Le cluster Kafka stocke des flux de données dans des catégories appelées «topics »

Qui utilise Kafka ?

Compatibilité

- .NET, Java, Node.js,
- Swift, PHP, Python,
- C/C++, Python, Go, Ruby, Rust, Storm, Perl
- …
- https://cwiki.apache.org/confluence/display/KAFKA/Clients

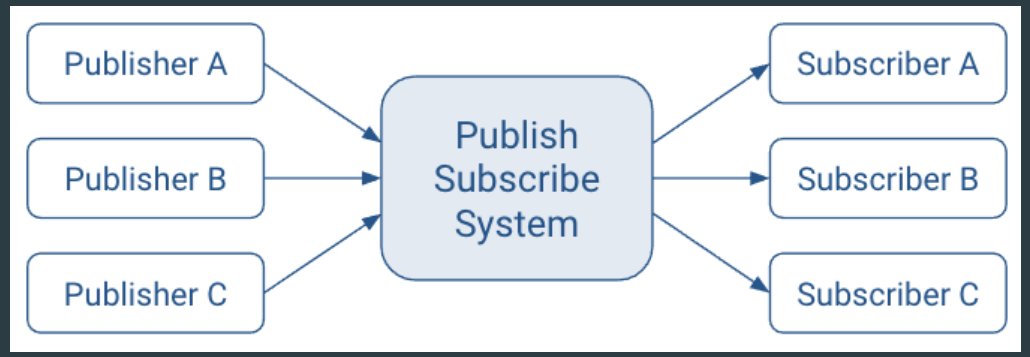
2) Fondamentaux

Fondamentaux

- Kafka contient 4 APIs:
  - Producer API
    - Permet à une application de publier un stream de records sur un ou plusieurs topics
  - Consumer API
    - Permet à une application de souscrire à un ou plusieurs topics ce qui lui permet de traiter les records reçus
  - Streams API
    - Permet de transférer en modifiant ou non le record d'un topic à un autre
  - Connector API
    - Permet de créer un producer ou consumer réutilisable et configurable. Cela nous permet de choisir le(s) topic(s) à utiliser en lançant le connector.

Fondamentaux
Publish-Subscribe

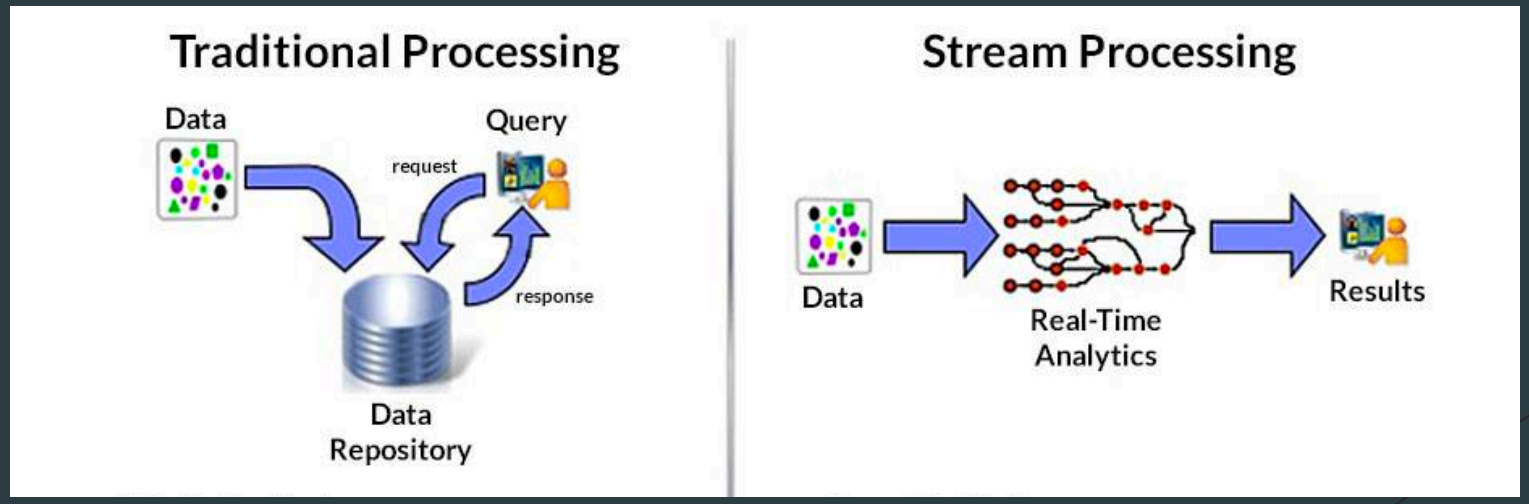
- Un système de message (publish – subscribe) idéal a :
  - Observations illimitées: un nouveau subscriber A1 pourrait lire les messages du publisher A à tout moment
  - Rétention de message : aucun message n'est perdu
  - Pas de nombre maximum de messages stockés
  - Jamais de coupure
  - Scaling illimité: nous pouvons mettre autant de subscriber et de publisher que nous voulons afin de réduire les latences en cas de masse de données



Fondamentaux
Event Stream

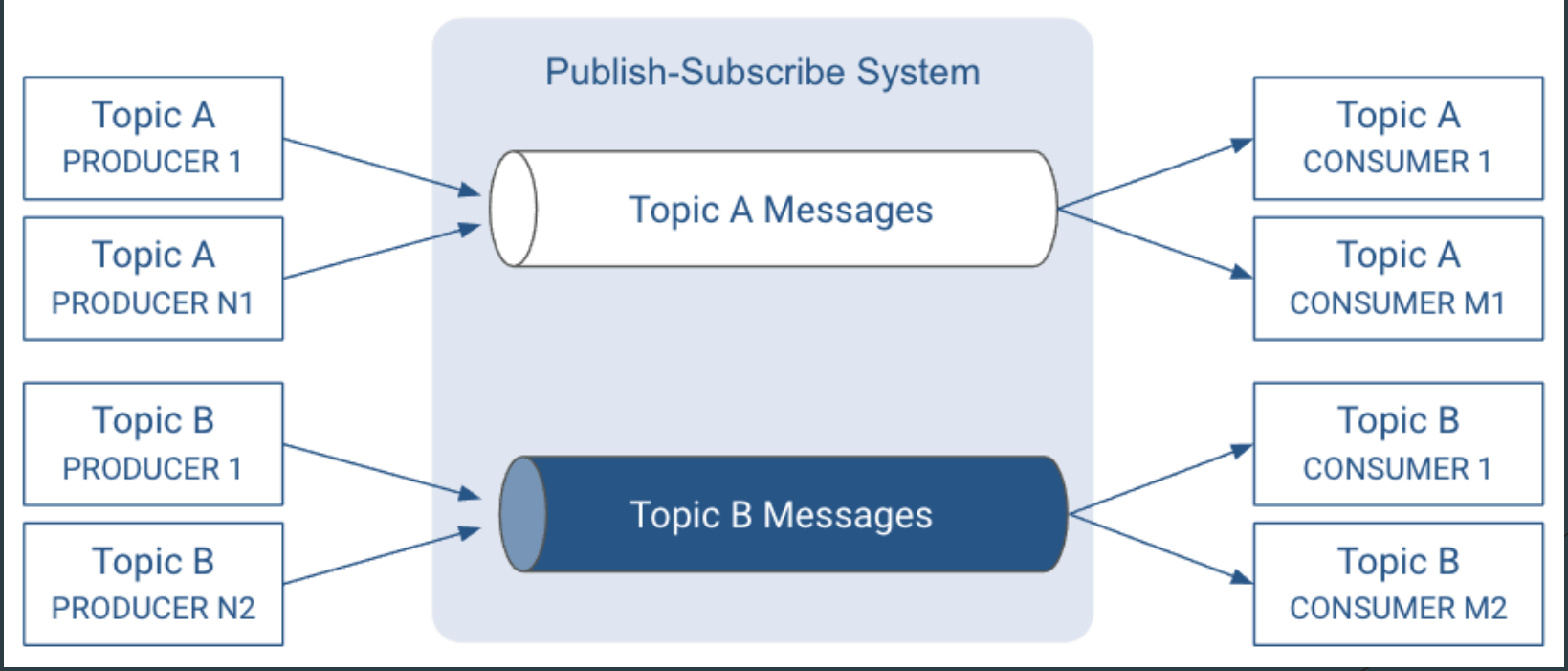
- Ordonné
- Immutable
- Rejouable

- Unidirectionnel / pas de réponse



Fondamentaux
Topics

- Un topic est une catégorie de message. Cela permet d'avoir plusieurs queues et ne de pas surcharger de messages les consumers qui ne sont intéressé que par une catégorie de messages.

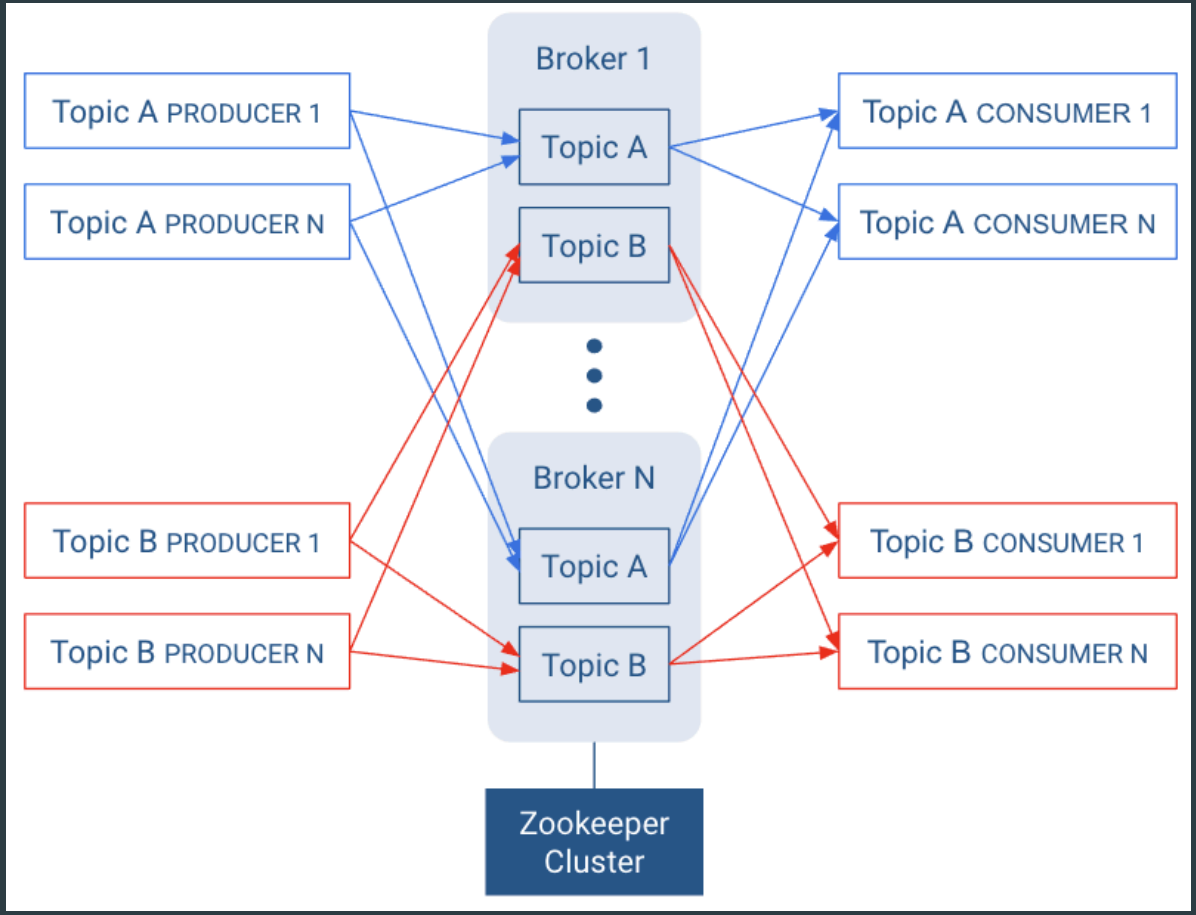


Fondamentaux
Brockers

- Chaque host faisant partie du cluster Kafka font tourné serveur appelé « brocker » qui va s'occupe de stocker les messages envoyés aux topics et de servir les requêtes des consumers

- Si un brocker venait à tomber, Kafka fait de son mieux pour re-dispatcher les messages non traités à un autre brocker
  - Jamais de coupure
- Tous les brockers communiques avec un Zookeeper pour coordonner ceux-ci
  - Scaling illimité
- Les topics sont répliqués sur chaque brocker
  - Rétention de message

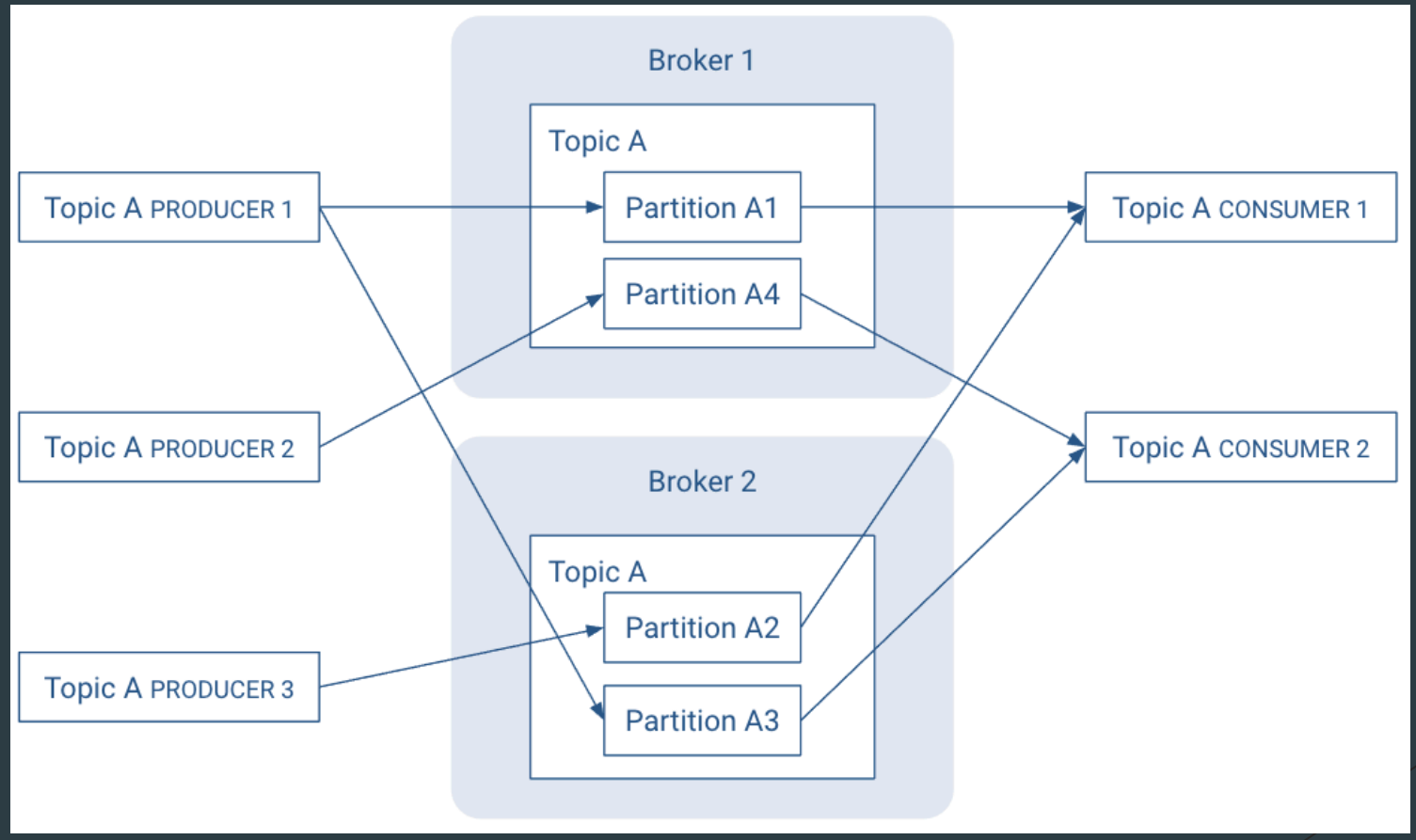
Fondamentaux
Brockers



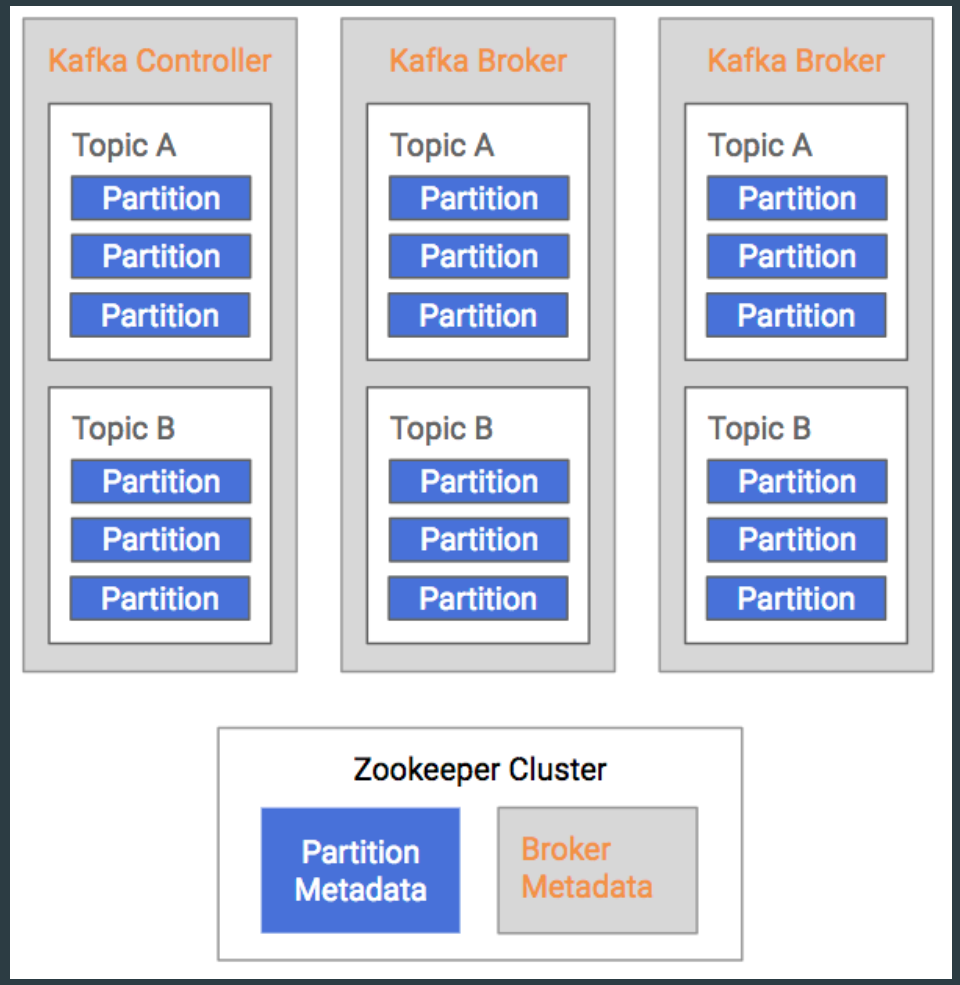
Fondamentaux
Partitions

- Plutôt que de stocker tous les records d'un topic dans un seul log (fichier), Kafka fait une découpe en partitions
- Chaque topic a un leader de partition. Si le facteur de réplication est configuré à plus que 1, la partition existera donc sur plusieurs serveurs (les followers)
- Chaque client Kafka (producer ou consumer) ne communique qu'avec le leader de la partition. Les autres n'existe qu'en tant que redondance ou failover
- Ce sont les followers qui sont responsables de copier les nouveaux records (messages) depuis le leader
  - Si le nombre de brockers (M) est plus grand que le nombre de facteurs de réplication (N) => M < N : alors chaque brocker aura une partie des partitions
  - Si le nombre de brockers (M) est égal au nombre de facteurs de réplication (N) => M = N : chaque brocker contiendra une copie complète des partitions

Fondamentaux
Partitions



Fondamentaux
Vue d'ensemble



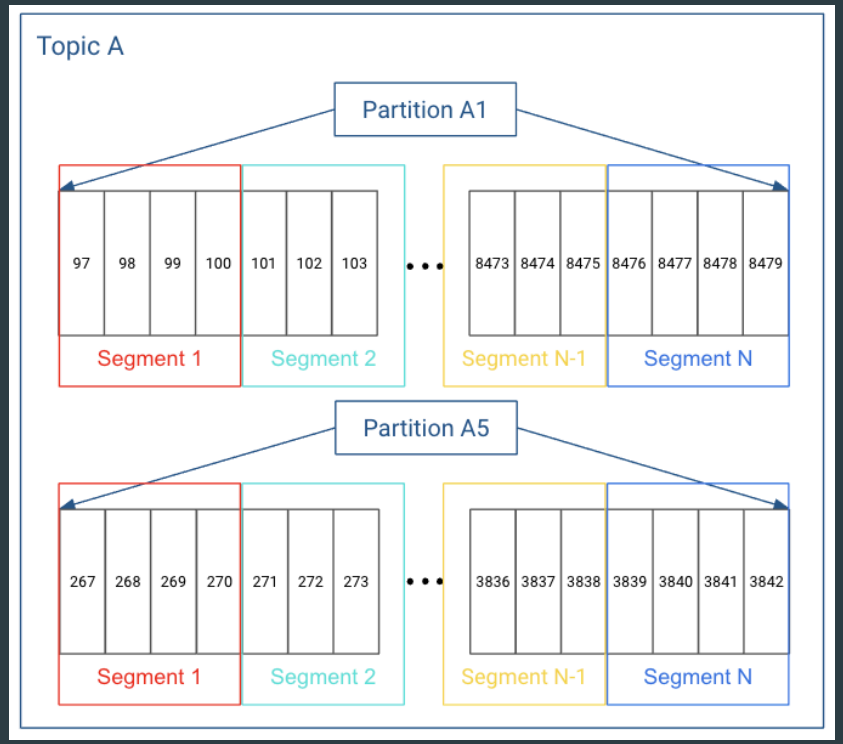
Fondamentaux
Assignation et ordre des records

- Par défaut l'assignation / ordonnancement est faite en Round-Robin, ce qui ne garantit évidemment pas que l'ordre des messages sera conservé car ils pourront être traités dans plusieurs partitions
- Kafka garantit cependant l'ordre des records dans une même partition ainsi que dans tous ces réplicats

- Si l'ordre à de l'importance, il faudra que tous les records liés se retrouvent dans la même partition. Cela peut se faire de deux façons
  - Spécifier la partition à utiliser dans le record
  - Utilise une clé d'assignement dans le record
    - Utiliser la même clé pour plusieurs records assure que ces records seront disponibles dans la même partition

Fondamentaux
Segments

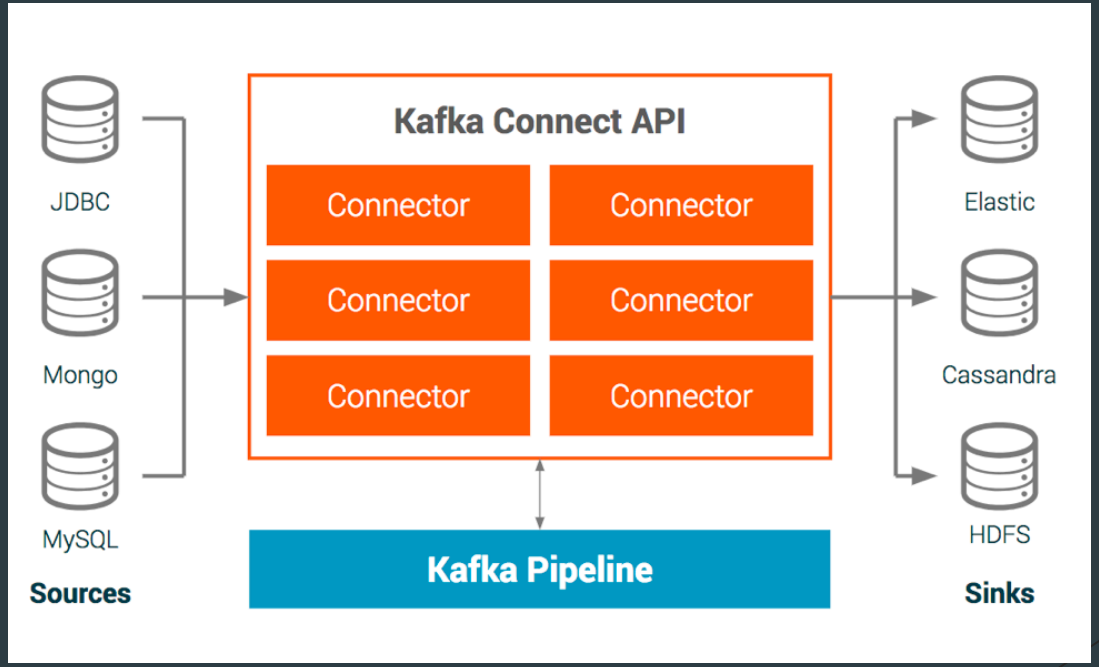
- Chaque partition est également éclatée en segments afin de limiter la taille des fichiers (exemple: 1GB maximum)



3) Confluent Hub

Confluent Hub

- https://www.confluent.io/hub/
- App Store pour Kafka
- Un ensemble de connectors partagés par la communauté



4) Kafka vs RabbitMQ

Kafka vs RabbitMQ

	Kafka	RabbitMQ
Distribution / Parallelisme	Oui	Non
Haute disponibilité	Oui (mais meilleure)	Oui
Performance	10 nœuds[1]	Plus de 30 nœuds[1]
Réplication	Automatique	Configurable
Multi subscriber	Oui	Non[2]
Ordre des messages	Oui	Non
Protocols de messages supportés	Types de messages primitifs (int, string…)	AMQP, STOMP, MQTT, HTTP
Durée de vie d'un message	Illimité	Une seule lecture

1. Nombre de nœuds nécessaire pour gérer plusieurs millions de requêtes par seconde
2. Une fois le message lu, il est supprimé et n'est plus jamais disponible

Kafka vs RabbitMQ

	Kafka	RabbitMQ
Ack	Oui[1]	Oui
Routage flexible	Non	Oui[2]
Priorité sur les messages	Non[3]	Oui
Monitoring	3rd party	Built-in
Support de transaction (rollback)	Oui	Oui

1. Acknowledgement manuel possible en désactivant « autoCommitOffset »
2. Il est possible de router les message par exemple via une regexp
3. Il n'est pas impossible, mais difficile à mettre en place à l'aide des clées

Kafka vs RabbitMQ
Contextes d'utilisation

- Kafka
  - Durabilité des messages
  - Plusieurs consumers pour un message
  - Débit élevé
  - Stream processing
  - Réplication
  - Haute disponibilité
  - Ordre des messages
- RabbitMQ
  - Routage flexible
  - Priorité
  - Utilisation d'un protocole de message standard

5) Azure services

Azure services

- Azure Event Hub = Kafka
  - Azure Event Hub supporte HTTP et AMQP alors que Kafka ne supporte que HTTP
  - Azure Event Hub applique une limitation en entrée/sortie, mais pas Kafka
  - Azure Event Hub oblige une sécurité par SAS alors que Kafka permet (optionnellement) une sécurité en SASL
  - Azure Event Hub limite la taille des messages à 1MB alors qu'il n'y a pas de limites pour Kafka
  - Le temps de rétention maximum sur Azure Event Hub est de 7 jours alors que illimité dans Kafka
- Event Hubs for Apache Kafka
  - Il existe un end-point qui permet aux clients de communiquer avec le protocole de Kafka et donc de créer des topics Kafka

Azure services

- Azure Service Bus ~ RabbitMQ
  - Classements
  - Détection de doublons
  - Détection de cohérence
  - FIFO
  - Routage
  - Filtrage

  - A favoriser par rapport à l'Azure Event Hub pour des données sensibles

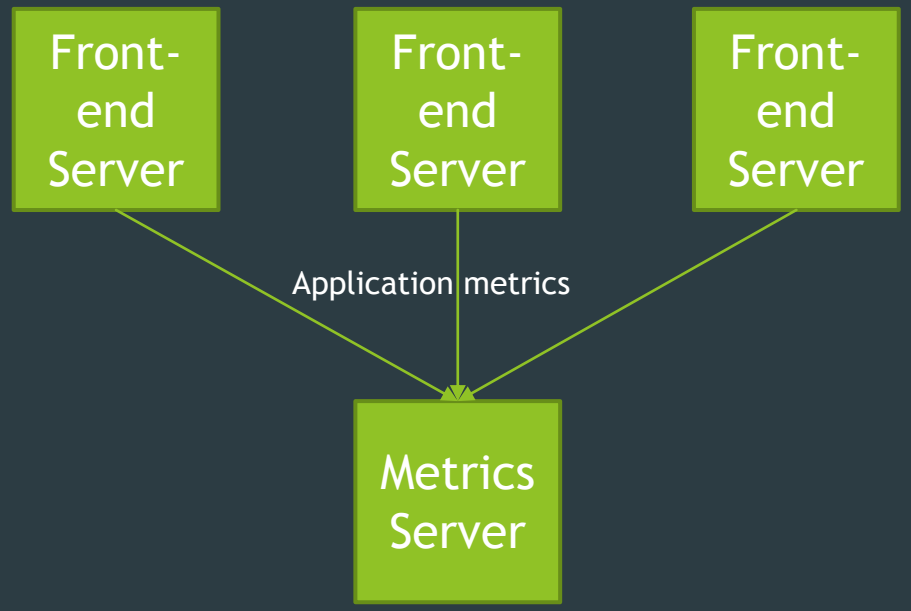
Azure services

- Azure Event Grid
  - Dynamiquement scalable
  - Economique
  - Serverless
  - Pas un Data Pipeline
    - Contient uniquement les informations nécessaires telles que type d'évènement et id de l'objet modifié
    - Récupération de l'objet modifié via un autre mécanisme

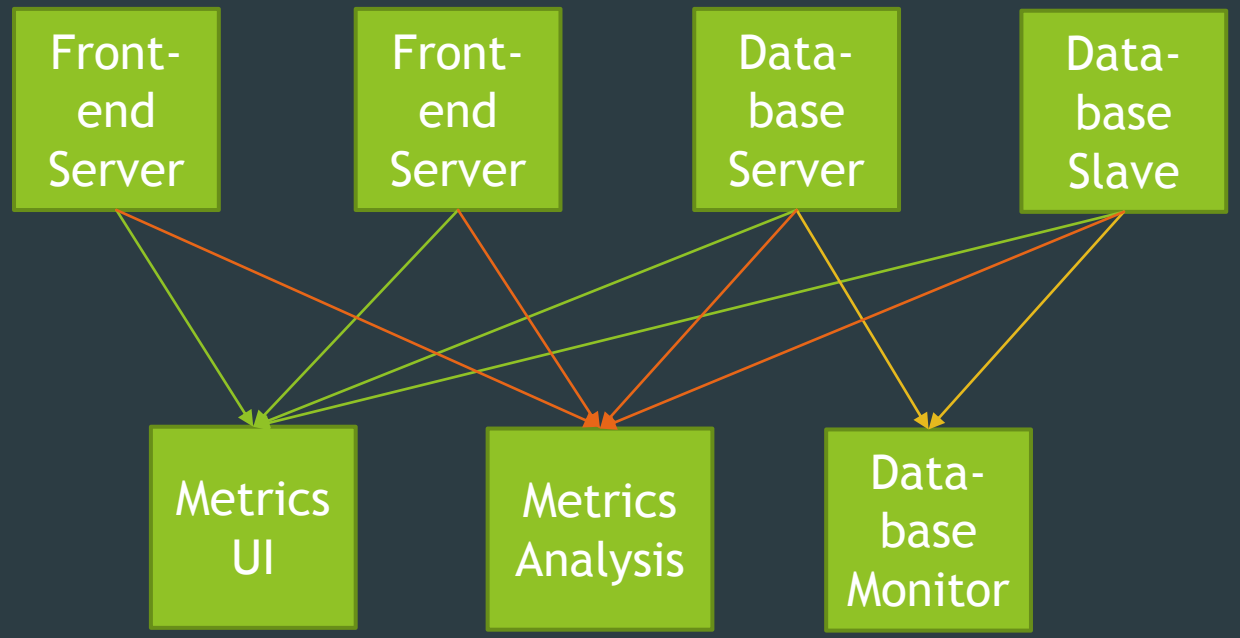
	Objectif	Type	Quand utiliser
Event Grid	Programmation réactive	Distribution d'événements (discrets)	Réagir aux changements d'état
Event Hubs	Pipeline de Big Data	Streaming d'événements (série)	Données de télémétrie et streaming de données distribuées
Service Bus	Messages importants de l'entreprise	Message	Traitement des commandes et transactions financières

5) Architecture micro-services

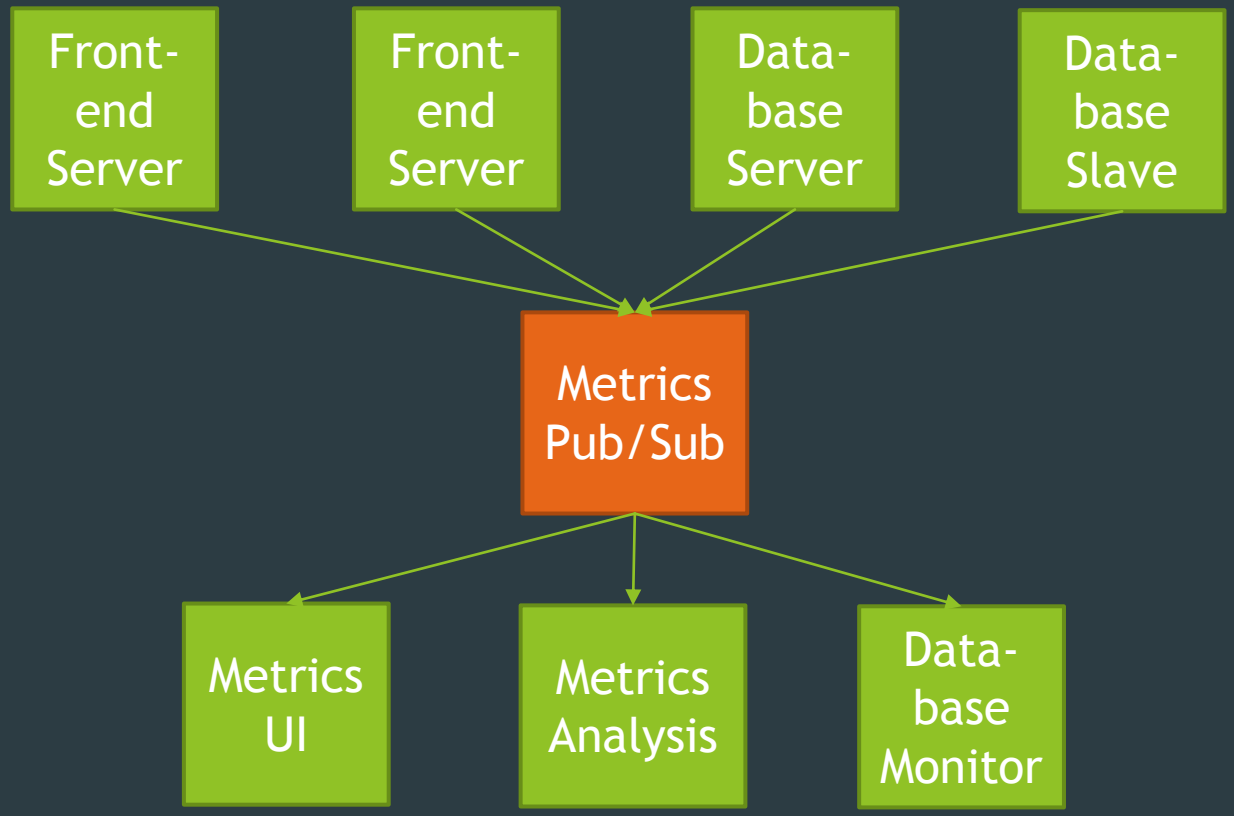
Architecture micro-services



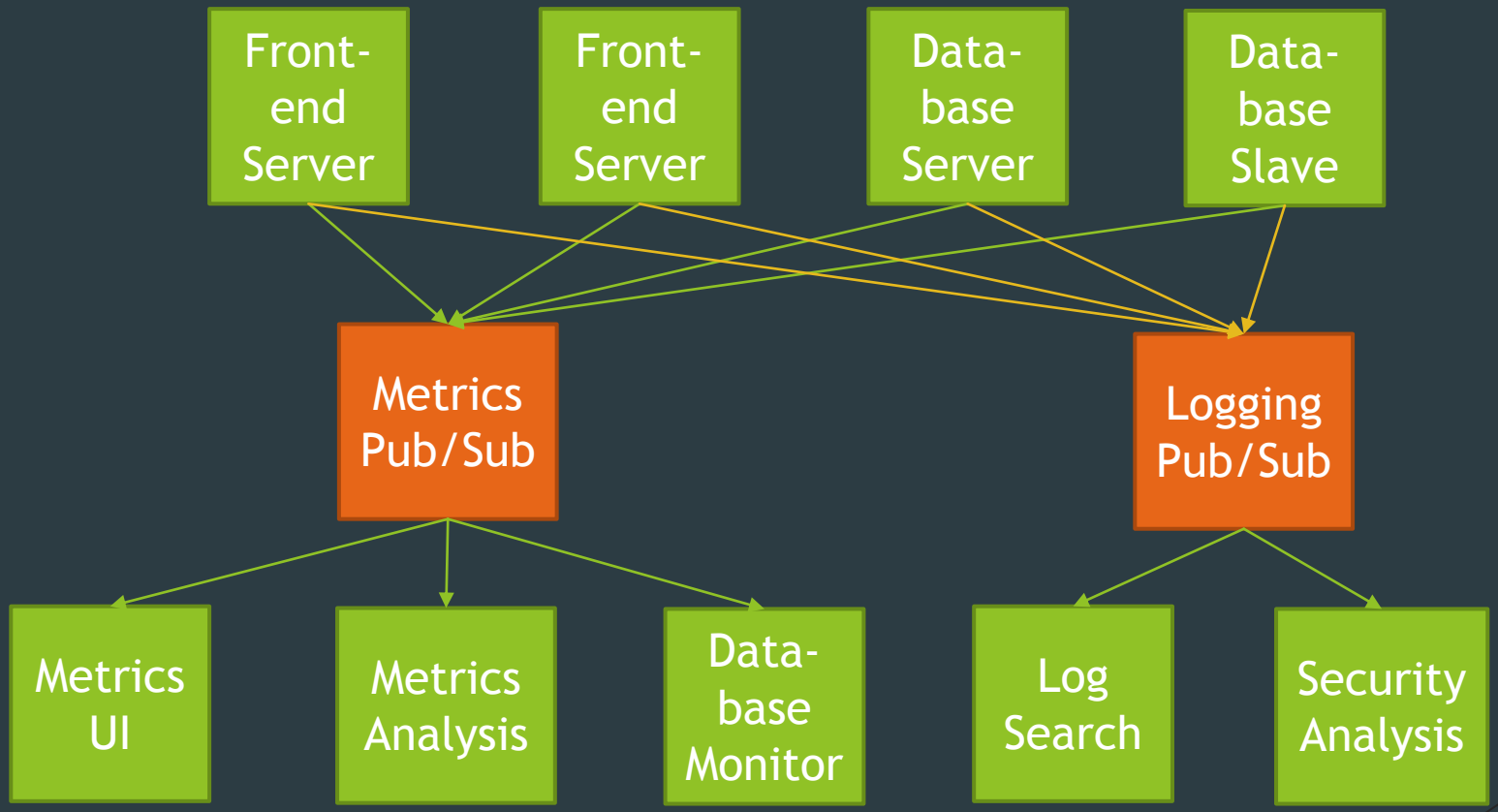
Architecture micro-services



Architecture micro-services



Architecture micro-services

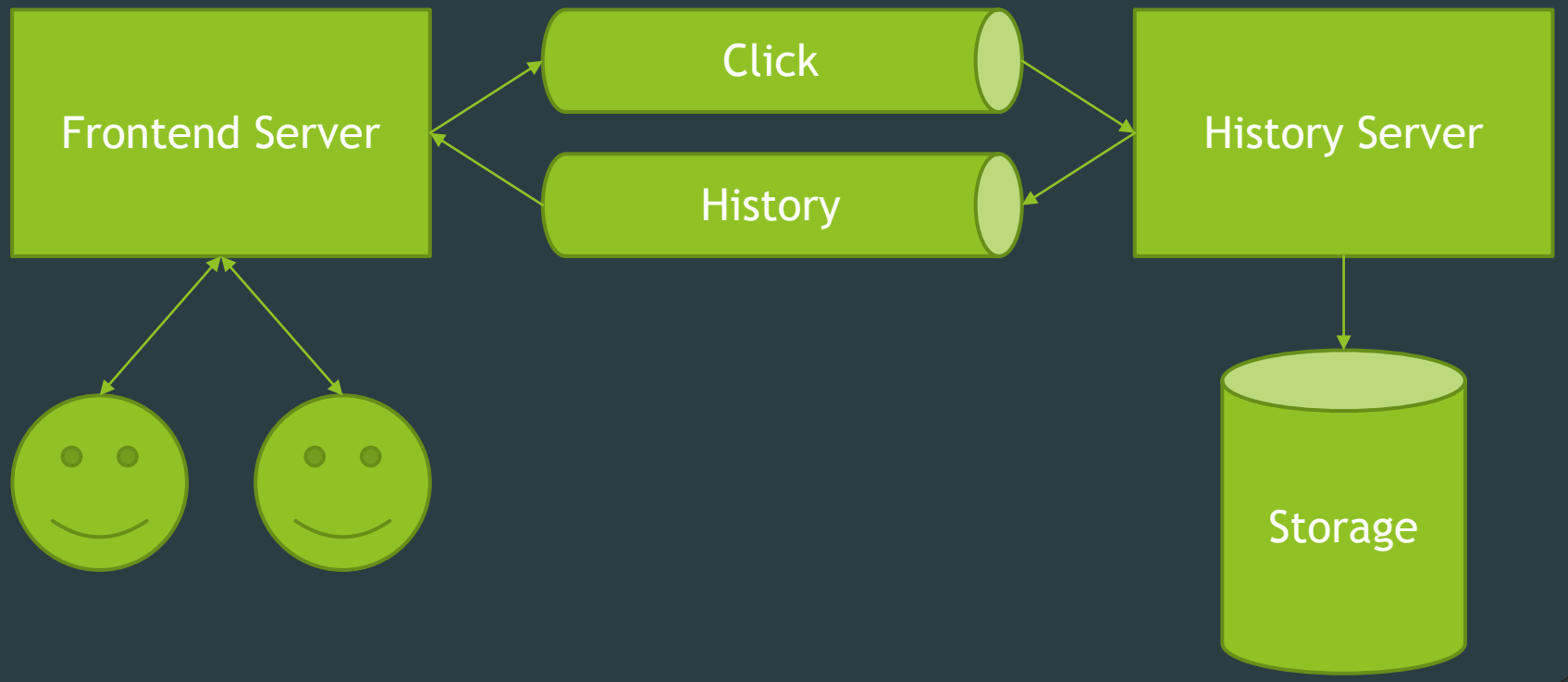


Architecture micro-services
Points d'attention

- Pas d'option « Exactly-once » pour .NET pour le moment
  - A vous de gérer le fait de déjà avoir reçu l'information
- Acks:
  - Par défaut il commit un offset représentant une liste de messages
  - Possibilité pour prendre la main:
    - Mettre l'option « enable.auto.commit=false » et commiter manuellement après le traitement du message
    - Mettre l'option « enable.auto.offset.store=false » et laisser « enable.auto.commit=true ». Il faudra alors mettre à jour manuellement le « offset_store »

6) Démo

Démo



Références

- https://www.cloudera.com/documentation/enterprise/latest/topics/kafka.html
- https://www.confluent.io/hub/
- https://itnext.io/kafka-vs-rabbitmq-f5abc02e3912
- https://github.com/confluentinc/confluent-kafka-dotnet/issues/523
- https://github.com/confluentinc/confluent-kafka-dotnet/issues/31